Memo

To: *Record*
From: *John Gipson*
Date: *January 21, 1999*
Re: *User_Partial, User_Program*

This memo describes how to use the user_partial feature of solve. It also describes how to use a special application of the general purpose feature user_program.

Introduction

When I first came to Goddard I thought that it would be nice to be able to estimate arbitrary parameters in a straightforward manner. For example, you might think that VLBI is sensitive to atmospheric pressure gradients. Wouldn't it be nice to be able to put in the partials and estimate the coeffiecients? Prior to the advent of user_partial the only way to do this was to make extensive modifications of solve and run with a special purpose version of solve. User_partial gives the researcher the ability to estimate arbitrary parameters as long as he can write a routine to calculate the partial derivatives of the delay and rate with respect to the partials.

In its raw form, user_partial involves a fair amount of bookkeeping. You open up various files, calculate partials, and write out the results to these files, close them and do cleanup. Since most of the bookkeeping stays the same regardless of the partial you are estimating, I wrote some template routines which isolate the bookkeeping to a single subroutine. This subroutine is called, oddly enough, *userpart.* This routine is paired with another routine that does the calculation. For example, the file *uen.f* calculates the UEN derivatives. All you need to do is to compile and link the two routines together to get a valid *user_partial* program.

After I began using *user_partial* for a while I found that I would repeatedly encounter the following. I would estimate some parameters in a global solution, for example Love numbers. I would then want to apply these parameters in another solution. This can be done using the *user_program* feature of solve. Actually *user_program* can do much more than this, but this is what I will focus on. The previously estimated parameters were being used as calibrations in the new solution. To apply them as calibrations you need to 1) Calculate the derivative of the delay with respect to the partials; 2.) Multiply the partial by the estimate; and 3.) Sum the result over all the parameters and subtract from the delay.

Similarly to *user_partials*, there is a fair amount of bookkeeping involved in user_program. Because of this I isolated the bookkeeping to a separate routine called *usercal.* Since the calculation of the derivatives is common to both user_partials and the calibration mode of user_program, I thought that it would be nice to have this calculation done by the same routine for both programs.

User_program/user_cal Program structure.

This section describes my templates for *user_program/user_cal*. Here *user_cal* is the *user_program* feature of solve used to calibrate the data. I will illustrate this by talking about a program to estimate the UEN components, or to calibrate the data by shifting the UEN components.

userpart. This program, contained in file usepart.f, does all of the bookkeeping for *user_partial*. It calls two user-supplied subroutines. The first subroutine is called during initialization, and it returns a list of partials we wish to estimate and their names. The calling format is:

```
call partial_names(lnames,num_partials,max_partials)

On entry:
        integer*4 max_partials--maximum number of allowed partials
On exit:
        integer*4 num_partials--number of partials we will estimate
        character*22 lnames(max_partials)  names of partials we estimate.
```

The partial names can be arbitrary at this stage, but presumably will have some mnemonic value. userpart then reads in the obs file, and once for each observation calls a subroutine to compute the partials. The calling format of this routine is:

```
call compute_partials(lnames,partial, num_partials)

On entry
        integer*4 num_partials    -- number of partials we are estimating.
        character*22 lnames(max_partials)  names of partials we estimate.
On exit
        double precision partial(2,num_partials)  delay and rate partials for each parameter.
```

In principle compute_partial doesn't need to use any of these variables as long as it returns the correct partial derivatives. For example, these partials could be hard wired. In practise what I have found best is for compute_partials to parse the character array to determine what partial to calculate.

usercal. This program, contained in file usercal.f, does all of the bookkeeping for user_program. It calls two user-supplied subroutines. The first routine is called during initilization. This routine returns a list of partial names and values. It is called:

```
call call_names_and_values(lnames,values,num_partials,max_partials)

On entry:
        integer*4 max_partials--maximum number of allowed partials
On exit:
        integer*4 num_partials--number of partials we will estimate
        character*22 lnames(max_partials)  names of partials we estimate.
        double precision values(max_partials)  values of the parameters in the same units
        estimated by user_partial.
```

User_cal then opens the obs file, and cycles through it, observation by observation. For each observation usercal then calls the compute_partials routine described above. After the call it then does the following (schematically):

```
delta_delay=0.
delta_rate=0.
do ipart=1,num_partials
   delta_delay=delta_delay+values(ipart)*partials(1,ipart)
   delta_rate=delta_rate+values(ipart)*partials(2,ipart)
end do.
```

The numbers delta_delay and delta_rate are the delay and rate calibration. These could be applied directly to the theoretical delay. I have found it better to put these in an unused calibration slot, say Shapiro, and then turn on this slot in *solve.* The advantage of this latter approach is that if you apply the calibration directly to the theoretical delay, every time you call usercal, the delay will change. This is not in a big problem in batch solutions, where you process each arc only once. But in interatctive solutions you tend to process each arc several times as you refine the solution. If you apply the correction as a calibration, you don't have to wory about repeadetly applying the same calibration.

Details of Use in Solve

USER_PARTIAL. This features allows the researcher to solve for arbitrary parameters as long as they can program the partials for the parameters. In batch mode it is invoked by including the line
   USER_PARTIALS complete_path_name
in the $SETUP part of the control file. It can be invoked interactively by using the "<" key in OPTIN.

USER_PROGRAM. This feature allows the researcher to apply an arbitrary calibration to the data. Actually, you can do anything you want to the data. In batch mode it is invoked by including the line
   USER_PROGRAM complete_path_name
in the $SETUP part of the control file. It can be invoked interactively by using the "&" key in OPTIN.

USER_BUFFER. Sometimes it is useful to pass information to USER_PROGRAM. This can be done by including the line:
   USER_BUFFER "ascii string"
in the $SETUP part of the control file. The ascii string can be arbitrary. I frequently use it to pass a file name which contains more information to the USER_PROGRAM. In interactive mode you are automatically prompted for a USER_BUFFER string when you run USER_PROGRAM.

Example Programs

Available via anonymous ftp is an example user_partial/user_cal program. This is in directory: gemini://pub/misc/jmg/uen_part. This contains 6 files:

userpart.f   Driver program to take care of user_partial bookkeeping.
usercal.f   Driver program to take care of bookkeeping for user_cal.
uen.f   Routines which compute the partials.

kdebug.f -- This checks the environment to see if debug mode is turned on.
makefile -- this will make two files: /home/tmp/UENPART, /home/tmp/UENCAL.
pos.cal -- this contains UEN calibrations for the data. Since it is a calibration, the new position estimate is the same as the original with these values subtracted from them.

UENPART will estimate the UEN components of the stations. The way it is written it will estimate the UEN positions of the first (N-1) stations in an N station experiment.

UENCAL calibrates the data so that the UP station position changes. As it is currently written, this puts the calibration in the Shapiro slot, so that this must be turned on for the calibration to work. Actually, it puts it in slot 10 of the "calibb" array, which is currently Shapiro. If this changes, for example, with CALC9, then you should put it in another unused slot.

The amount of changes is currently read in from the file POS.CAL. The name of this file could easily be specified at runtime using the "USER_BUFFER" feature of solve. One way of doing so is indicated in the subroutine "cal_names_and_values" in uen.f.

Debugging Information

If the environment variables "debug" or "DEBUG" are set "yes" or "on" or "true" (all case insensitive) then usercal and userpart will print out debugging info to two files: usercal.dbg and userpart.dbg in the users current directory. This list the partials estimated, their valuations and the calibrations. For exactly what they print out, see the source file.